# Dyn-FO: A Parallel, Dynamic Complexity Class

(Preliminary Version)

**Sushant Patnaik**
University of Massachusetts, Amherst
patnaik@cs.umass.edu

**Neil Immerman***
University of Massachusetts, Amherst
immerman@cs.umass.edu

## Abstract

Traditionally, computational complexity has considered only static problems. Classical Complexity Classes such as NC, P, NP, and PSPACE are defined in terms of the complexity of checking – upon presentation of an entire input – whether the input satisfies a certain property.

For many, if not most, applications of computers including: databases, text editors, program development, it is more appropriate to model the process as a dynamic one. There is a fairly large object being worked on over a period of time. The object is repeatedly modified by users and computations are performed.

Thus a dynamic algorithm for a certain class of queries is one that can maintain an input object, e.g. a database, and process changes to the database as well as answering queries about the current database.

Here, we introduce the complexity class, Dynamic First-Order Logic (Dyn-FO). This is the class of properties S, for which there is an algorithm that can perform inserts, deletes and queries from S, such that each unit insert, delete, or query is first-order computable. This corresponds to the sets of properties that can be maintained and queried in first-order logic, i.e. relational calculus, on a relational database.

We investigate the complexity class Dyn-FO. We show that many interesting properties are in Dyn-FO including, among others, graph connectivity, k-edge connectivity, and the computation of minimum spanning trees. Furthermore, we show that several NP complete optimization problems admit approximation algorithms in Dyn-FO. Note that none of these problems is in static FO, and this fact has been used to justify increasing the power of query languages beyond first-order. It is thus striking that these problems are indeed dynamic first-order, and thus, were computable in first-order database languages all along.

We also define "bounded expansion reductions" which honor dynamic complexity classes. We prove that certain standard complete problems for static complexity classes, such as AGAP for P remain complete via these new reductions. On the other hand, we prove that other such problems including GAP for NL and 1GAP for L are no longer complete via bounded expansion reductions. Furthermore, we show that a version of AGAP called $AGAP^+$ is not in Dyn-FO unless all of P is contained in parallel linear time.

Our results shed light on some of the interesting differences between static and dynamic complexity.

*Research partially supported by NSF grant CCR-9207797.



SIGMOD/PODS 94 - 5/94 Minneapolis, Minnesota USA
© 1994 ACM 0-89791-639-5/94/0005..$3.50

## 1 Introduction

Traditional complexity classes are not completely appropriate for database systems. Unfortunately, appropriate Database Complexity Classes have not yet been defined. This paper makes a step towards correcting this situation.

In our view, the main two differences between database complexity and traditional complexity are:

1. Databases are **dynamic**. The work to be done consists of a long sequence of small updates and queries to a large database. Each update and query should be performed very quickly in comparison to the size of the database.

2. Computations on databases are for the most part **disk access bound.** The cost of computing a request is usually tied closely to the number of disk pages that must be read or written to fulfill the request.

Of course, a significant percentage of all uses of computers have the above two features. In this paper we focus on the first issue. Dynamic Complexity is quite relevant in most day to day tasks. For example: Texing a file, Compiling a program, Processing a visual scene, Performing a complicated calculation in Mathematica, etc. Yet an adequate theory of dynamic complexity is lacking. (Recently, there have been some significant contributions in this direction, e.g. [MSVT93]. Note that dynamic complexity is different although somewhat related to On-line complexity which is receiving a great deal of attention lately.)

We will define the complexity class Dyn-FO to be the set of dynamic problems that can be expressed in first-order logic. What this means is that we maintain a database of relevant information so that the action invoked by each insert, delete, and query is first-order expressible. This is very natural in the database setting. In fact, Dyn-FO is really the set of queries that are computable in a traditional first-order query language.

Many interesting queries such as connectivity for undirected graphs are not first-order when considered as static queries. This has led to much work on database query languages such as Datalog that are more expressive than first-order logic.

We show the surprising fact that a wealth of problems, including connectivity, are in Dyn-FO. Thus, considered as dynamic problems – and that is what database problems are – these problems are already first-order computable. The problems we show to be in Dyn-FO include: reachability in undirected graphs, maintaining minimum spanning forest, $k$-edge connectivity and bipartiteness. All regular languages are shown to be in Dyn-FO. We even show that decent approximation algorithms for several NP complete problems are in Dyn-FO. Previously, Dong and Su, in [DS93] have shown that reachability in directed, acyclic graphs and in function graphs is in Dyn-FO. The static versions of all these problems are, of course, not first-order.

### 1.1 Related Work

In [DST93], Dong, Su and Topor consider the incremental evaluation problem for Datalog queries, namely, repeatedly evaluating the same Datalog query to a database that is being updated between successive query requests. They define a first order incremental evaluation system (FOIES), with respect to a given Datalog query, where the incremental evaluation is carried out by a non-recursive Datalog program. They point out that non-recursive Datalog programs are much better than recursive ones using elaborate data structures for database applications, since they reduce the number of relational join operations. Our approach is similar to their approach in the sense that both store derived relations for reuse after updates. However, Dyn-FO is a general complexity class and it involves both insertions and deletions. Monotone Dyn-FO (which allows only insertions) is equivalent to FOIES. Defining the arity of a Dyn-FO expression as the arity of the auxiliary (non-input) relations used in the first order logic formulae for handling updates, we see that the notion of *space-free* FOIES is related to devising minimum arity Dyn-FO[1] expressions, in the sense that problems in *space-free* FOIES, by definition, have minimum arity Dyn-FO expressions. The central theme in the incremental approach, namely, to use the difference between successive database states and the answer to the query in one state to reduce the cost of evaluating the query in the next state, plays a role in maintaining materialized views upon updates ([J92],[GMS93]), and in integrity constraint simplification ([LST87], [N82]). In these studies, the authors investigate how to maintain first-order definable views efficiently under updates to the underlying database. In our framework, we can interpret their approaches as determining ways to implement fast Dyn-TIME solutions for a subclass of Dyn-FO. For example, [GMS93] show that a very restricted subclass of Dyn-FO is in Dyn-TIME[1].

The design of dynamic algorithms is an active field. See, for example, [E*92], [R92], [CT91], [F83], amongst others. This paper is also informed by [MSVT93] which does some of the ground work for a complexity theory of dynamic complexity.

[1] We shall use Dyn-FO interchangably to denote a class of decision problems and a language

This paper is organized as follows. In section 2, we begin with some background on Descriptive Complexity. In section 3, for any static complexity class $\mathcal{C}$, we define the corresponding dynamic class, Dyn-$\mathcal{C}$. The class Dyn-FO is the case we emphasize. Then, in section 4, we present several of the above mentioned Dyn-FO algorithms. In section 5, we describe and investigate reductions honoring dynamic complexity. Finally, we conclude with a list of a few of the many directions that we feel should be pursued.

## 2 Descriptive Complexity: Background and Definitions

Our approach depends on insights and intuitions developed via an alternate notion of complexity, Descriptive Complexity, that ensues from considering the power of a language needed to *express* some property, in contrast to the traditional approach of measuring the complexity of *checking* a property. This was first studied by Fagin in [Fa74] where he gave a characterization of non-deterministic polynomial time (NP) as the set of properties expressible in second-order existential logic. Later, Gurevich, Immerman, Vardi, and others showed that each natural complexity class has a natural characterization from a descriptive point of view, [G83, I87, I89, I91, Var].

It typically turned out that that "natural" complexity classes have "natural" descriptive characterizations. For example, space corresponds to number of variables; and, parallel time is linearly related to quantifier-depth. Sequential time on the other hand does not seem to have a natural descriptive characterization. We like to think of this as yet another indication that sequential time is not a natural notion, simply an artifact of the so-called "Von-Neumann bottleneck". As another example, the class P, consisting of those problems that can be performed in a polynomial amount of work, has an extremely natural characterization as (FO + LFP) – first-order logic closed under the ability to make inductive defintions.

It is reassuring that our notions of naturalness in logic correspond so nicely with naturalness in complexity theory. In the present work we venture into the terrain of dynamic complexity. What is natural is not yet clear. We use the intuitions gained from the descriptive approach to aid us in our search.

In this section we recall the notation of Descriptive Complexity. See [I89] for a survey and [IL89] for an extensive discussion of the reductions we use here including first-order projections.

We will code all inputs as finite logical structures, i.e., relational databases. A *vocabulary* $\tau = \langle R_1^{a_1} \ldots R_s^{a_s}, c_1, \ldots, c_t \rangle$ is a tuple of input relation and constant symbols. Let STRUC[$\tau$] denote the set of all finite structures of vocabulary $\tau$. We define a complexity theoretic *problem* to be any subset $S \subseteq$ STRUC[$\tau$] for some $\tau$.

For any vocabulary $\tau$ there is a corresponding first-order language $\mathcal{L}(\tau)$ built up from the symbols of $\tau$ and the logical relation symbols, $=$ and $\leq$, (which refers to a total ordering on the universe), using logical connectives: $\wedge, \vee, \neg$, variables: $x, y, z, \ldots$, and quantifiers: $\forall, \exists$.

In traditional static complexity, the entire input structure $\mathcal{A}$ is fixed and we are interested in deciding whether $A \in S$ for a relevant property, $S$. In the dynamic case, the structure changes over time. The actions we have in mind are a sequence of insertions and deletions of tuples in the input relations. We will usually think of our dynamic structure, $\mathcal{A} = \langle \{0, 1, \ldots n-1\}, R_1, \ldots, R_s, c_1, \ldots, c_t \rangle$, as having a fixed size potential universe, $|\mathcal{A}| = \{0, 1, \ldots n-1\}$, and a unary relation $R_1$, specifying the elements in the active domain. The initial structure of size $n$ for this vocabulary will be taken to be $\mathcal{A}_0^n = \langle \{0, 1, \ldots n-1\}, \{0\}, \emptyset, \ldots, \emptyset, 0, \ldots, 0 \rangle$, having $R_1 = \{0\}$ indicating that the single element 0 is in the active domain and all the other relations are empty.

### 2.1 First-Order Reductions

Here we briefly describe first-order reductions, a natural way of reducing one problem to another in the Descriptive context. First-order reductions are used in Section 5 to build new reductions that honor dynamic complexity. Furthermore, reductions are used in Section 3 as a motivation for the definition of Dynamic Complexity. More information about first-order reductions can be found in [IL89].

**Definition 2.1** Let $S \subseteq$ STRUCT[$\sigma$] and $T \subseteq$ STRUCT[$\tau$] be two problems and $\tau = \langle R_1^{a_1} \ldots R_t^{a_t} \rangle$.

Let $I = \{\varphi_0 ... \varphi_t\}$ be a set of $t+1$ formulas where the free variables of $\varphi_i$ are a subset of $\{u_1, ... u_{k \cdot a_i}\}$. $I$ induces a mapping

$$I : \text{STRUCT}[\sigma] \longrightarrow \text{STRUCT}[\tau]$$

such that if $A \in \text{STRUCT}[\sigma]$ then

$$I(A) = \langle I(\mathcal{U}), R_1, ..., R_t \rangle$$

such that

$$\begin{aligned} I(\mathcal{U}) &= \{i \in \{0, \ldots, \mathbf{n}^k - 1\} \mid \mathcal{A} \models \varphi_0(i)\} \\ &= \{\langle u_1 u_2 \cdots u_k \rangle \in \mathcal{U}^k \mid \mathcal{A} \models \varphi_0(\bar{u})\} \end{aligned}$$

and the instantiation of each $R_i$ is determined by $\varphi_i$ in the following way

$$(I(A) \models R_i(\langle u_1, ..., u_k \rangle \cdots \langle u_{k \cdot (a_i - 1)}, ..., u_{k \cdot a_i} \rangle))$$

$$\Leftrightarrow \ (A \models \varphi_i(u_1, ..., u_{k \cdot a_i}))$$

where $\langle u_1, ..., u_k \rangle$ denotes $u_1 n^{k-1} + u_2 n^{k-2} + \cdots + u^k$ element of $\{0, ..., n^k - 1\}$.

If $I$ is a many-one reduction from $S$ to $T$, i.e,

$$A \in S \ \Leftrightarrow \ I(A) \in T$$

Then $I$ is called a *k-ary first order interpretation of S to T.*

**Definition 2.2** A first order projection (fop) is a first-order interpretation such that each of the formulae $\varphi_0, ..., \varphi_t$ is of the form

$$\alpha_0 \vee (\alpha_1 \wedge \lambda_1) \vee \cdots \vee (\alpha_r \wedge \lambda_r) \qquad (1)$$

where the $\alpha_i$'s are mutually exclusive formulas in which no relation symbols occur, and each $\lambda_i$ is a literal (i.e.) a relation symbol or its negation. Since $I(A) \models R_i(\langle u_1, \ldots, u_k \rangle \cdots \langle u_{k \cdot (a_i - 1)}, \ldots, u_{k \cdot a_i} \rangle)$ when $A \models \varphi_i(u_1, \ldots, u_{k \cdot a_i})$, then $R_i(\langle u_1, ..., u_k \rangle \cdots \langle u_{k \cdot (a_i - 1)}, \cdots, u_{k \cdot a_i} \rangle)$ would be satisfiable in $I(A)$ when $\alpha_0$ is true or when $\alpha_j$ is true for some $j$ and the corresponding $\lambda_s$ is true as well. Thus, every bit in the representation on $I(A)$ is determined by at most one bit in the representation of $A$.

# 3 Dynamic Complexity Classes

We think of an implementation of a problem $S \subseteq \text{STRUC}[\sigma]$ as a mapping, $I$ from $\text{STRUC}[\sigma]$ to $\text{STRUC}[\tau]$ where $T \subseteq \text{STRUC}[\tau]$ is an easier problem. The map $I$ should be a many-one reduction from $S$ to $T$ meaning that any structure $\mathcal{A}$ has the property $S$ iff $I(\mathcal{A})$ has the property $T$. (Actually, in our definition below, the mapping $I$ will map a sequence of inserts and deletes $\bar{r}$ to a structure. In the interesting special case when $I(\bar{r})$ depends only on the corresponding structure $\mathcal{A}$ and not which sequence of inserts and deletes created it, we call $I$ *memoryless.*)

We are thinking and talking about a structure $\mathcal{A} \in \text{STRUC}[\sigma]$, but the structure that we actually have in memory and disk and are manipulating is $I(\mathcal{A}) \in \text{STRUC}[\tau]$. In this way, each insert or delete on $\mathcal{A}$ is interpreted as a corresponding series of actions on $I(\mathcal{A})$. The fact that $I$ is a many-one reduction insures that the query asking whether $\mathcal{A} \in S$ can be answered according as whether $I(\mathcal{A}) \in T$.

Next we give the formal definition of dynamic complexity classes. The issue is that the structure $I(\mathcal{A})$ can be updated efficiently in response to any insert or delete to $\mathcal{A}$. In particular, if $T \in$ FO and all such inserts and deletes are first-order computable, then $S \in$ Dyn-FO.

## 3.1 Definition of Dyn-$\mathcal{C}$

For any complexity class $\mathcal{C}$ we define its dynamic version, Dyn-$\mathcal{C}$, as follows. Let $\sigma = \langle R_1^{a_1} \ldots R_s^{a_s}, c_1, \ldots, c_t \rangle$ be a vocabulary and let $S \subseteq \text{STRUC}[\sigma]$ be any problem. Let $\mathcal{R}_{n,\sigma} = \{\text{ins}(i, \bar{a}), \text{del}(i, \bar{a}), \text{set}(j, a) \mid 1 \leq i \leq s, \bar{a} \in \{0, \ldots, n-1\}^{a_i}, 1 \leq j \leq t\}$ be the set be possible requests to insert tuple $\bar{a}$ into the relation $R_i$, delete tuple $\bar{a}$ from relation $R_i$, or set constant $c_j$ to $a$.

Let $\text{eval}_{n,\sigma} : \mathcal{R}^\star_{n,\sigma} \rightarrow \text{STRUC}[\sigma]$ be the naturally defined evaluation of a sequence of requests, initialized by $\text{eval}_{n,\sigma}(\emptyset) = \mathcal{A}_0^n$.

Define, $S \in$ Dyn-$\mathcal{C}$ iff there exists another problem $T \subset \text{STRUC}[\tau]$ such that $T \in \mathcal{C}$ and there exist maps,

$$f : \mathcal{R}^\star_{n,\sigma} \rightarrow \text{STRUC}[\tau];$$

$$g : \text{STRUC}[\tau] \times \mathcal{R}_{n,\sigma} \rightarrow \text{STRUC}[\tau]$$

satisfying the following properties.

1. For all $\bar{r} \in \mathcal{R}^\star_{n,\sigma}$, $(\text{eval}_{n,\sigma}(\bar{r}) \in S) \ \Leftrightarrow \ (f(\bar{r}) \in T)$
2. For all $s \in \mathcal{R}_{n,\sigma}$, and $\bar{r} \in \mathcal{R}^\star_{n,\sigma}$, $f(\text{eval}_{n,\sigma}(\bar{r}s)) = g(f(\text{eval}_{n,\sigma}(\bar{r})), s)$

3. $\|f(\bar{r})\| = \|\text{eval}_{n,\sigma}(\bar{r})\|^{O(1)}$, where for any structure, $\mathcal{A}$, $\|\mathcal{A}\|$ denotes the size of $\mathcal{A}$@. [2]

4. The functions $g$ and the initial structure $f(\emptyset)$ are computable in complexity $\mathcal{C}$, (as a function of $n$).

We will say that the above map $f$ is *memoryless* if the value of $f(\bar{r})$ depends only on $\text{eval}_{n,\sigma}(\bar{r})$.

In the above, if only inserts and queries are considered, i.e., no deletes, then we get the class Dyn$_s$-$\mathcal{C}$, the semi-dynamic version of $\mathcal{C}$. One can also consider amortized versions of these two classes. Furthermore, there are some cases where we would like extra, but polynomial, precomputation to compute the initial structure $f(\emptyset)$. If we relax condition (4) in this way, then the resulting class is called Dyn-$\mathcal{C}^+$ – Dyn-$\mathcal{C}$ with polynomial precomputation.

We have thus defined the dynamic complexity classes Dyn-$\mathcal{C}$ for any static class, $\mathcal{C}$. Two particularly interesting examples are Dyn-FO, and Dyn-TIME[$t(n)$] for $t(n) \in o(n)$ where the later is the set of problems computable dynamically on a RAM in time $t(n)$.

## 4 Problems in Dyn-FO

Let graph reachability denote the following problem: given a graph, G, and vertices $x, y$, determine if there is a path from $x$ to $y$ in G. We shall use 1GAP, UGAP, GAP (acyclic), respectively, to denote graph reachability on directed graphs with out-degree at most 1, undirected graphs and acyclic directed graphs where the inserts preserve acyclicity. It is well known that the graph reachability problem is not first-order expressible and this has often been used as a justification for using database query languages more powerful than FO [CH82]. Thus, the following two theorems are striking.

**Theorem 4.1** *UGAP is in* Dyn-FO.

**Proof** We maintain a spanning forest of the underlying graph via relations, $\text{F}(x, y)$ and $\text{PV}(x, y, u)$ and the input relation, E. $\text{F}(x, y)$ means that the edge $(x, y)$ is in the current spanning forest. $\text{PV}(x, y, u)$ means that there is a (unique) path in the forest from $x$ to $y$ via vertex $u$. The vertex, $u$, may be one of the endpoints. So, for example, if $\text{F}(x, y)$ is true, then so are $\text{P}(x, y, x)$ and $\text{P}(x, y, y)$. We maintain teh undirected nature of the graph by interpreting insert(E,$a, b$) or delete(E,$a, b$) to do the operation on both $(a, b)$ and $(b, a)$.

The first operation of interest is: **Insert(E,**$a, b$**)**: We denote the updated relations as E′, F′ and PV′. In the sequel, we shall use $\text{P}(x, y)$ to abbreviate $\text{P}(x, y, x)$, and $\text{Eq}(x, y, c, d)$ to abbreviate the formula,

$$((x = c \wedge y = d) \vee (x = d \wedge y = c)).$$

Maintaining the input edge relation is trivial:

$$E'(x, y) \equiv E(x, y) \vee Eq(x, y, a, b)$$

The edges in the forest remain unchanged, if vertices, $a$ and $b$, were already in the same connected component. Otherwise, the only new forest edge is $(a, b)$.

$$F'(x, y) \equiv F(x, y) \vee (Eq(x, y, a, b) \wedge \neg P(a, b))$$

Now all that remains is to compute PV′. The latter changes iff edge $(a, b)$ connects two formerly disconnected trees, and in this case, the new tuples added are the pairs of vertices and tree (or, forest) edges from the two disjoint trees (one containing $a$ and the other, $b$) that are merged.

$$\begin{aligned}&\text{PV}'(x, y, z) \equiv \\ &\text{PV}(x, y, z) \vee (\text{Eq}(x, y, a, b) \wedge (z = a \vee z = b)) \vee \\ &\neg\text{P}(x, y) \wedge (\exists u, v)\ \text{Eq}(u, v, a, b) \wedge P(x, u) \wedge P(v, y) \\ &\qquad\qquad \wedge(z = a \vee z = b \vee\ \text{PV}(x, u, z) \\ &\qquad\qquad \vee\ \text{PV}(v, y, z))\end{aligned}$$

**Delete(E,**$a, b$**)**: If edge $(a, b)$ is not in the forest ($\neg$ $\text{F}(a, b)$), then the updated relations are unchanged, except that $\text{E}'(a, b)$ is set to false. Otherwise, we first identify the vertices of the two trees in the forest created by the deletion, and then we pick an edge, say $e$, out of all the edges (if any) that run between the two trees and *insert* $e$ into the forest, updating the relations, PV and F, appropriately.

We define a temporary relation T to denote the PV relation after $(a, b)$ is deleted, before the new edge, $e$, is inserted. $\text{T}(x, y, z) \equiv$

$$\text{PV}(x, y, z) \wedge \neg(\text{PV}(x, y, a) \wedge\ \text{PV}(x, y, b))$$

[2] This expects that the complexity class $\mathcal{C}$ is closed under polynomial increases in the input size. For more restricted classes $\mathcal{C}$, such as linear time, we insist that $\|f(\bar{r})\| = O(\|\text{eval}_{n,\sigma}(\bar{r})\|)$.

Using T, we then pick the new edge that must be added to the spanning forest. New$(x,y)$ is true if and only if edge $(x,y)$ is the minimum [3] edge that connects the two disconnected components:

$$\begin{aligned}\text{New}(x,y) \equiv\ & \text{E}(x,y) \wedge \text{T}(a,x,a) \wedge \text{T}(b,y,b) \wedge \\ & (\forall u,v)(\text{E}(u,v) \wedge \text{T}(a,u,a) \\ & \wedge \text{T}(b,v,b)) \rightarrow (x < u \\ & \vee (x = u \wedge y \leq v))\end{aligned}$$

E′, F′ and PV′ are then defined as follows:

$$\text{E}'(x,y) \equiv \text{E}(x,y) \wedge \neg\text{Eq}(x,y,a,b)$$

We remove $(a,b)$ from the forest and add the new edge.

$$\text{F}'(x,y) \equiv (\text{F}(x,y) \wedge \neg\text{Eq}(x,y,a,b)) \vee \text{New}(x,y) \vee \text{New}(y,x)$$

The paths in the forest, from $x$ to $y$ via $z$, that did not pass through $a$ and $b$, are valid.
Also, new paths have to be added as a result of the insertion of a new edge in the forest.

$$\begin{aligned}\text{PV}'(x,y,z) \equiv\ & \text{T}(x,y,z) \vee \\ & [(\exists u,v)(\text{New}(u,v) \vee \text{New}(v,u)) \wedge \text{T}(x,u,x) \\ & \wedge \text{T}(y,v,y) \wedge (\text{T}(x,u,z) \wedge \text{T}(y,v,z))]\end{aligned}$$

The proof of correctness is straightforward. ∎

We give a new Dyn-FO algorithm for the following result.

**Theorem 4.2 ([DS93])** *1GAP and GAP (acyclic) are in* Dyn-FO.

**Proof** 1GAP follows easily from UGAP. For the GAP (acyclic) case, the inserts are assumed to always preserve acyclicity. We maintain the path relation, P$(x,y)$ which means that there is a path from $x$ to $y$ in the graph.

**Insert(E,$a,b$)**:

$$\text{P}'(x,y) \equiv \text{P}(x,y) \vee (\text{P}(x,a) \wedge \text{P}(b,y))$$

**Delete(E,$a,b$)**:

$$\begin{aligned}\text{P}'(x,y) \equiv\ & \text{P}(x,y) \wedge [\neg(\text{P}(x,a) \wedge \text{P}(x,y)) \vee \\ & (\exists u,v)\ \text{P}(x,u) \wedge \text{P}(u,a) \wedge \text{E}(u,v) \wedge \neg\text{P}(v,a) \\ & \wedge \text{P}(v,y) \wedge (v \neq b \vee u \neq a)]\end{aligned}$$

In the case where there is a path from $x$ to $y$ using the edge $(a,b)$, consider any path not using this edge. Let $u$ be the last vertex along this path from which $a$ is reachable. Note that $u \neq y$ because the graph was acyclic even before the deletion of edge, $(a,b)$. Thus, the edge, $(u,v)$, described in the above formula must exist and acyclicity insures that the path $x \rightarrow u \rightarrow v \rightarrow y$ does not involve the edge, $(a,b)$. The proof of correctness is straightforward. ∎

Let TR denote the following problem: For G, a directed acyclic graph, recall that the *Transitive Reduction*, TR(G), is the minimal subgraph of G having the same transitive closure as G.

**Corollary 4.3** *Transitive Reduction for directed acyclic graphs is in memoryless* Dyn-FO.

**Proof** We maintain the path relation, P, as in Theorem 4.2.
**Insert(E,$(a,b)$)**: If P$(a,b)$ already holds, then there is no change. Otherwise, we have to remove some edges from TR:

$$\begin{aligned}\text{TR}'(x,y) \equiv\ & (\text{TR}(x,y) \wedge \text{P}(a,b)) \\ & \vee[\neg\text{P}(a,b) \wedge (x = a \wedge y = b) \vee [\text{TR}(x,y) \\ & \wedge\neg(\text{P}(x,a) \wedge \text{P}(b,y))]]\end{aligned}$$

**Delete(E,$(a,b)$)**: We have to determine the new edges that might be added in TR. Then, New$(x,y)$ is defined as follows:

$$\begin{aligned}\text{New}(x,y) \equiv\ & \text{E}(x,y) \wedge \neg\text{TR}(x,y) \wedge \text{P}(x,a) \wedge \\ & \text{P}(b,y) \wedge [(\forall z)(\text{P}(x,z) \wedge \text{P}(z,y)) \rightarrow \\ & \text{P}(b,z) \vee \text{P}(z,a)]\end{aligned}$$

$$\begin{aligned}&\text{TR}'(x,y) \equiv \\ &(\text{TR}(x,y) \wedge \neg(x = a \wedge y = b)) \vee \text{New}(x,y)\end{aligned}$$

The proof of correctness is straightforward. ∎

Let Minimum Spanning Forest denote the following problem: given an undirected graph G with weights on the edges, determine the minimum weighted spanning forest of G. We may assume w. l. o. g. that weights are distinct and hence, the minimum spanning tree is unique (since otherwise, ties can be broken by imposing a lexicographic ordering on the edges along with its weights).

**Theorem 4.4** *Minimum Spanning Forest is in* Dyn-FO.

**Proof** The general idea is to maintain the forest edges and non-forest edges dynamically and to maintain the relations $\mathrm{PV}(x, y, e)$ and $\mathrm{F}(x, y)$ as in the case of UGAP. Let $\mathrm{W}(a, b)$ denote the weight of edge $(a, b)$. The difference from UGAP is that we have to maintain the minimum weighted forest. That changes our update procedures in the following way.

Deletion of an edge, say $(a, b)$, is handled as follows. We determine, using PV, all the vertices that can be reached from $a$ in the tree and all those that can be reached from $b$. These give the vertices in the two trees that the original tree splits into. Then, instead of choosing the lexicographically first non-forest edge that reconnects the two pieces, we choose the *minimum weight* such edge, and insert it. PV is updated accordingly to reflect the merging of two disconnected trees into one.

When an edge,say $(a, b)$ is inserted, we determine if there exists a path between $a$ and $b$. If there is no path, then $(a, b)$ merges two trees into one, and PV is updated as before for UGAP. Otherwise, using PV, we can determine the forest-edges that appear in the unique path in the forest between $b$ and $a$, and check to see if weight of the new edge, $(a, b)$ is less than the weight of any of these edges. If not, then $(a, b)$ is not a forest edge and nothing changes. Otherwise, let $(c, d)$ be the maximum weight edge on the path from $a$ to $b$. We make $\mathrm{F}'(c, d)$ false and $\mathrm{F}'(a, b)$ true and update PV accordingly. The proof of correctness is straightforward. ∎

We next show that similar simple algorithms exist for Bipartiteness, Edge Connectivity, Least Common Ancestor queries (in rooted trees), Maximal Matching and Maximal Independent Set (in bounded degree graphs). These latter two have no known sub-linear time fully dynamic solutions.

**Theorem 4.5** ([**PI94**]) *Let $k$ be a fixed constant.* Dyn-FO *algorithms exist for the following problems:*

1. *Bipartiteness,*
2. *$k$-Edge Connectivity,*
3. *Maximal Matching in undirected graphs,*
4. *Least Common Ancestor in rooted trees.*

**Proof** Omitted. ∎

It is an interesting phenomenon that constant-approximations to certain NP complete problems can be maintained by FO relational formulae. Consider the well-known $\frac{1}{2}$-approximate solution to the triangular Traveling Salesman Problem ($\Delta$TSP). Modifying the Dyn-FO algorithm for maintaining a minimum spanning tree to maintain an Euler tour of the tree simultaneously enables us to show that

**Theorem 4.6** ([**P94**]) *A $\frac{1}{2}$-approximation to $\Delta$TSP is in Dyn-FO.*

**Proof** Omitted. ∎

**Corollary 4.7** *A $\frac{1}{2}$-approximation to $\Delta$TSP is in Dyn-TIME[$\sqrt{n}\log n$].*

**Proof** Follows from the result in [E*92] that minimum spanning tree is in Dyn-TIME[$\sqrt{n}\log n$], and the proposition in [P94] that Euler tour of a spanning tree can be maintained in constant time. ∎

A $\frac{1}{2}$-approximate (greedy) solution to Minimum Node Cover and any Maximal Independent Set for bounded (say, $k$) degree graphs (and hence, a $\frac{k}{k+1}$-approximate solution) can also be maintained in Dyn-FO. It is well known that MAX-SNP, an approximation class introduced by Papadimitriou and Yannakakis in the framework of first-order logic in [PY88], can be approximated to within some fixed $\epsilon$ in polynomial time, but this algorithm unfortunately does not lend itself to a Dyn-FO solution. It would be interesting to determine whether there exists a generic Dyn-FO solution for the entire MAX-SNP class that is $\epsilon$-approximate, for some fixed constant $\epsilon$, depending only on the MAX-SNP problem as usual (and not the size of the input to the problem). However, it is possible to show that approximate solutions to a subclass of MAX-SNP, that includes MAX SAT and MAX CUT, can be maintained in Dyn-FO, in fact, in Dyn-TIME[1] ([P94]).

## 5 Dynamic Reductions

Now we define appropriate reductions for comparing dynamic complexity classes. For these classes first order reductions are too powerful. We restrict them by imposing the following *expansion* property, cf. [STV93] for a similar restriction.

**Definition 5.1** *Bounded expansion FO reductions* (bfo) are FO interpretations (Definition 2.1) such that each bit of the input structure affects at most a constant number of bits in the output structure. If $S$ is reducible to $T$ via bounded-expansion, first-order reductions, we write $S \leq_{\text{bfo}} T$.

In a bfo reduction, the initial structure, $\mathcal{A}_{0,n}$ must be mapped to a structure with only a bounded number of tuples in its relations. If this condition is relaxed, then we get bfo$^+$ reductions.

Recall that a projection (Definition 2.2) has the property that each bit of the output is determined by at most one bit of the input. The bounded expansion property is sort of the converse of this. Due to lack of space we just state or briefly sketch the relevant properties of bfo's. The full details may be found in [PI94].

**Proposition 5.2** *For any problems $S$ and $T$, if $T \in$* Dyn-FO *and $S \leq_{\text{bfo}} T$, then $S \in$* Dyn-FO. *If $T \in$* Dyn-FO$^+$ *and $S \leq_{bfo^+} T$, then $S \in$* Dyn-FO$^+$.

**Proof** Given a bfo reduction from S to T, any change to an input, $\mathcal{A}$, for S corresponds in a given first-order way to a bounded number of changes to I($\mathcal{A}$). Since T $\in$ Dyn-FO, so is S. ∎

It turns out that most natural reductions for P and NP complete problems are either bounded expansion, or can be easily modified to be so (see [P94], [M94]). For classes contained in NP, we can show, for example,

**Proposition 5.3** *AGAP remains complete for P via bfo$^+$ reductions.*

**Proof** In [I87], it is shown that CVAL (or, equivalently AGAP) is complete for P using first-order projections. The latter basically describe the circuit (or, graph) that encodes the polynomial sized computation tableau of a polytime Turing machine, where the input bits need be read only once, at the bottom level of the tableau. Hence, they are bounded expansion reductions. The precomputation is needed to construct the whole circuit. ∎

For any problem S, that is complete via $\leq^+_{\text{bfo}}$ reductions, there is another problem S$^+$ that is complete via $\leq_{\text{bfo}}$ reductions. Namely, in S$^+$, a portion of the structure is "understood" to be given by some fixed first-order formula, $\varphi_0$. For example, AGAP$^+$ is given as the set of "graph inputs" $H_n$ such that $H_n$ coupled with $\varphi_0(\mathcal{A}_{0,n})$ results in a graph in AGAP. Problems, S$^+$, thus, are represented by an immutable fixed, first-order formula, plus the usual (dynamic) input part.

**Lemma 5.4** *For any problem $S$, hard for a complexity class $C$ via bfo$^+$ reductions $S^+$ is hard for $C$ via $\leq_{\text{bfo}}$ reductions.*

An interesting corollary results, indicating that AGAP$^+$ is probably not in Dyn-FO. Let CRAM[$t(n)$] be the set of problems computable by uniform CRCW-PRAMS using polynomially much hardware. It is known that FO = CRAM[1]. It follows that:

**Proposition 5.5** Dyn-FO $\subseteq$ CRAM[$n$]. *Thus, if AGAP $\in$* Dyn-FO$^+$ *then $P \subseteq$* CRAM[$n$] *with polynomial time precomputation. If AGAP$^+$ $\in$* Dyn-FO, *then $P \subseteq$* CRAM[$n$].

Things are not as nice for lower complexity classes: L and NL. The reason is that bfo's preserve the number of times that the input is read. (We don't care about this for P and above as we may simply copy the input, reading it once.) We can thus prove that the standard complete problems for L and NL are not complete under bfo's:

**Theorem 5.6** *1GAP (GAP) is not complete for L (NL) via bounded expansion reductions with precomputation.*

**Proof** In [BRS91], the authors define the following problem: Let $\mathbf{F}_3$ denote the field with 3 elements. Let $n = 2^r$. Let A denote the

$2^r \times 2^r$ dimension Sylvester matrix, i.e. an entry $a_{ij}$ of A is 1 (in $\mathbf{F}_3$) if the dot product of binary vectors, $i$ and $j$, over $\mathbf{Z}_2$ is 0 and $-1$ (in $\mathbf{F}_3$), otherwise. Let

$$\begin{aligned} f(X,Y) &= 1, \text{if } X^T A Y = 0 \text{ (in } \mathbf{F}_3) \\ &= 0, \text{otherwise} \end{aligned}$$

where $A$ is the Sylvester matrix and $X$ and $Y$ are $n$ column input vectors over $\mathbf{F}_3$. Let $\mathrm{NBP}_k(f)$ denote the size of a non-deterministic branching program computing $f$ such that any input variable is read only $k$ times along any accepting path. It was shown by Borodin, Razborov and Smolensky that

**Fact 5.7 ([BRS91])** *$NBP_k(f) \geq exp(\Omega(\frac{n}{4^k k^3}))$.*

For a fixed $k$, let $k$-read only (NL) L denote the class of problems in (non-deterministic) Logspace such that each input bit is read only $k$ times (on any accepting computation). The following property of bounded expansion reductions is readily proved.

**Proposition 5.8** Let $S, T$ be problems. Let $\mathcal{C}$ be any complexity class that is closed with respect to FO reductions. Let $k, k'$ be fixed constants. If $S \leq_{bfo} T$ and $T \in k$-read only $\mathcal{C}$ then $S \in k'$-read only $\mathcal{C}$.

1GAP (GAP) is in $k$-read only L (NL), since any input bit is read at most once (on any accepting computation). Since NL is exactly the class of polynomial sized non-deterministic branching programs, by the above theorem, $f$ is not in $k$-read only NL, for any fixed constant $k$. But on the other hand, $f$ is clearly in L. Hence, assuming that 1GAP (or, GAP) is complete for L via $\leq_{bfo}$ reductions would imply that $f \leq_{bfo}$ 1GAP. Hence, it would follow from the above proposition that $f \in k$-read only L (NL), for some fixed $k$, which is a contradiction. ∎

In [STV93] the authors define a variant of GAP, COLORGAP as follows: given an acyclic directed graph, G = (V,E), edges colored 0 or 1, two distinguished vertices, $s$ and $t$, a partition of the vertices into equivalence classes, $V_1, \ldots, V_k$, and a bit vector $C[1, \ldots, k]$, determine if there exists a path in G from $s$ to $t$ such that all edges on the path directed out of vertices belonging to equivalence class $V_i$ have color $C_i$. Restating their result in our framework,

**Fact 5.9 ([STV93])** *COLORGAP is complete for NL via $\leq_{bfo^+}$ reductions.*

We can define a similar variant of the 1GAP problem, COLOR1GAP, as follows: given a directed graph, G = (V,E), with nodes of out-degree at most 2, and edges colored 0 or 1 such that if a vertex has out-degree 2 then the edges out of it have different colors, two distinguished vertices, $s$ and $t$, a partition of the vertices into equivalence classes, $V_1, \ldots, V_k$, and a bit vector $C[1, \ldots, k]$, determine if there exists a path in G from $s$ to $t$ such that all edges on the path directed out of vertices belonging to equivalence class $V_i$ have color $C_i$.

**Theorem 5.10** *COLOR1GAP is complete for L via $\leq_{bfo^+}$ reductions.*

**Proof** Given any logspace Turing machine, M, with a binary alphabet and an input $w$ (all 0's initially) of length $n$, we precompute the polynomial sized acyclic directed graph, G, where every vertex denotes a configuration of M, i.e. a Boolean encoding of the state, the working tape contents and the position of the input tape head and the working tape head, and the edges correspond to the moves of M, i.e. $E(x, y)$ iff configuration $x$ leads to configuration $y$ in one move of M. Initially, for any vertex, $x$, all the edges out of $x$ are colored 0, 1 according to the value of the input bit being read, and no two edges out of $x$ have the same color, since M is deterministic. Partition the vertices such that vertex, $x$, is in partition $i$ iff M reads the $i$-th input in the configuration corresponding to $x$, and the bit vector $C(i)$ equals the value of the $i$-th input bit. There is a path from the vertex corresponding to the initial configuration, $s$, to that of the final configuration, $f$, in G iff M accepts $w$.

The edges and vertices for the COLOR1GAP problem can be expressed by a FO interpretation (see, for example, [IL89]). The equivalence classes and the color vector (initially all 0's) are trivially expressed by FO interpretations. Now, whenever an input bit, say the $j$-th bit, flips, we just have to flip the corresponding color vector bit, $C(j)$. The reduction, therefore, has the bounded expansion property. ∎

## 6 Dyn-FO versus $NC^1$, L and NL

We have shown that 1GAP and GAP(acyclic) are in Dyn-FO. These problems are complete for L and NL respectively. However, the problems do not stay complete via bfo's and so it does not follow that NL, or even L, or even $NC^1$ is contained in Dyn-FO. We can prove the following, but most relations between Dyn-FO and static complexity classes are open.

**Theorem 6.1** *The following classes of problems are in* Dyn-FO*:*

1. *All regular languages,*
2. *$D^k$, the Dyck language on k parentheses, for any k.*

*The following classes of problems are in* Dyn-FO$^+$*:*

3. *k-read only L, for any k,*
4. *k-read only NL, for any k.*

**Proof**

1. Let $Q$ denote the state set of any DFA. The idea is to maintain the fixed sized mapping $Q \to Q$ that is induced by every symbol of the input word, $w$ ($|w| = n$) at the leaves of a balanced binary tree, and at every internal node the composition of the mappings of its children. The height of the tree is $\log n$. Since, the information stored at any node is only of constant size ($2|Q|$), on changing any symbol, say $w[i]$, a FO formula can guess the $O(\log n)$ bits along the unique path from leaf $i$ to the root and verify it against the values of the stored tree relation and then update it in constant time (by computing the new composition at the internal nodes or by a table look-up at the leaf).

2. We show the $D^2$ case. The theorem follows by an easy adaptation. $D^2$ can be parsed using the level trick: assign a level to each parenthesis starting at one and ignoring the differences in parenthesis type. The *level* of a parenthesis equals the number of left parentheses to its left (including it) minus the number of right parentheses strictly to its left. A right parenthesis matches a left one if it is the closest parenthesis to the right on the same level. A string is in $D^2$ iff all parentheses have a positive level and each left parenthesis has a matching right parenthesis of the same type.

   In [BC89], the authors showed that $D^2 \in TC^0$. We basically note that the relations they used can be updated in FO. Let
   LEFT(RIGHT)$(i) \equiv$ parenthesis at position $i$ is a left (right)type
   LEVEL$(i,l) \equiv$ parenthesis at position $i$ is at level $l$
   MATCH$(i,j) \equiv$ parenthesis at position $i$ matches parenthesis at $j$

   $$\text{D2} \equiv \text{input is in } D^2.$$

   LEFT, RIGHT, MATCH and D2 can be described by FO formulae. For example,

   $$D^2 \equiv (\forall i)(\exists l)(l > 0 \wedge \text{LEVEL}(i,l)) \wedge$$

   $$(\forall j)(\exists k)\text{LEFT}(j) \wedge \text{RIGHT}(k) \wedge \text{MATCH}(j,k)$$

   Let $\{\#x : f(x)\}$ denote the number of $x$'s such that $f(x)$ is true. LEVEL is then expressed as:

   $$\begin{aligned}\text{LEVEL}(i,l) \equiv (\exists y)(y = \#x : x \le i \\ \wedge \text{OPEN}(x)) \wedge \\ (\exists z)(z = \#x : x < i \wedge \\ \text{CLOSE}(x)) \\ \wedge y \ge z \wedge l = y - z\end{aligned}$$

   Clearly LEVEL is in DYN-FO i.e. LEVEL can be updated in FO: under an insert$(x,'[)$ operation, for example, we have for $i < x$, LEVEL$'(i,l)$ same as LEVEL$(i,l)$, for $i > x$, LEVEL$'(i,l)$ iff LEVEL$(i, l-1)$, and for $i = x$, LEVEL$'(x,l)$ iff for some $m$, LEVEL$(x-1,m)$ and ($(l = m-1$ and OPEN$(x-1))$ or (CLOSE$(x-1)$ and $l = m)$). The deletes and insert of a closed parenthesis can be handled similarly.

3. Given a logspace Turing machine, M, we use the standard polynomial (actually FO) precomputation (see, for example [I87], [P]) to build the acyclic directed graph, G, (with out-degree at most 1) corresponding to M's computation on blank input. Then, for any bit change in the input, by assumption, since it is read only constant number of times, only constantly many edges are altered in G. Thus, we can maintain the reachability on G in DYN-FO by composing

the DYN-FO algorithm for 1GAP constant number of times.

4. Given a non-deterministic logspace Turing machine, M, we use the standard polynomial (actually FO) precomputation (see, for example [I87], [P]) to build the acyclic directed graph, G, corresponding to M's computation on blank input. Then, for any bit change in the input, by assumption, since it is read only constant number of times, only constantly many edges are altered in G. Further we can assume the new edges that result are always such that acyclicity of G is preserved, since w.l.o.g. the computation graph can always be made cyclic. Thus, we can maintain the reachability information for G in DYN-FO by composing the DYN-FO algorithm for (acyclic) GAP constant number of times. ∎

Note that proofs of 3. and 4. above actually show that

**Theorem 6.2** *1. 1GAP is complete for k-read only L via $\leq_{bfo^+}$ reductions.*

*2. GAP is complete for k-read only NL via $\leq_{bfo^+}$ reductions.*

## 7 Conclusions

We have defined dynamic complexity classes, and their reductions. In particular, we have begun an investigation of the rich dynamic complexity class Dyn-FO. Much work remains to be done. We point towards a few of the many directions.

1. Does Dyn-FO contain any of the complexity classes: $TC^0$, $NC^1$, L, NL? We conjecture that $TC^0 \subseteq$ Dyn-FO and that AGAP $\notin$ Dyn-FO, thus P $\not\subseteq$ Dyn-FO.

2. Further work needs to be done concerning the role and value of precomputation in the dynamic setting.

3. Many of our constructions showing problems in Dyn-FO could be improved. For example, in Theorem 4.1, is UGAP $\in$ Dyn-FO using only relations of arity 2? The tradeoff of "number of extra bits saved" versus "savings in dynamic time" deserves a great deal of further study.

## 8 Acknowledgements

Thanks to Ron Fagin for helpful comments on a previous version of this paper.

## References

[ACF90] B. Alpern, L. Carter and E. Feig. Uniform memory hierarchies. In *Proceedings of the Thirty-first st Symposium on Foundations of Computer Science*, pages 600–608, 1990, IEEE.

[A*90] B. Alpern, R. Hoover, B. K. Rosen, P. F. Sweeney and F. Kenneth Zadeck. Incremental evaluation of computational circuits. In *Proceedings of the First Symposium on Discrete Algorithms* pages 32–42, 1990, ACM-SIAM.

[BC89] D. M. Barrington and J. C. Corbett. On the relative complexity of some languages in $NC^1$. In *Technical Report* pages 89–22, 1989, Department of Computer Science, University of Massachusetts, Amherst.

[BRS91] A. Borodin, R. Razborov and S. Smolensky. On lower bounds for read-$k$ times branching programs. In *Preprint*, 1991, Department of Computer Science, University of Toronto.

[CFI89] J. Cai, M. Fürer and N. Immerman. An optimal lower bound on the number of variables for graph identification. In *Proceedings of the Thirtieth Symposium on Foundations of Computer Science* pages 612–617, 1989, IEEE.

[CH82] A. Chandra and D. Harel, Structure and complexity of relational queries. In *Journal of Computer and System Sciences* **25** pages 99-128, 1982.

[CT91] R. Cohen and R. Tamassia. Dynamic expression trees and their applications. *Proceedings of the Second Annual SODA*, 1991, ACM-SIAM.

[DS93] G. Dong and J. Su. Incremental and decremental evaluation of transitive closure by first-order queries. In *Preprint*, 1993, Department of Computer Science, University of California, Santa Barbara.

[DST93] G. Dong, J. Su and R. Topor. First-order incremental evaluation of Datalog queries. In *Proceedings of the International Conference on Database Theory*, Lecture Notes in Computer Science 646, 1992, Springer Verlag.

[E*92] D. Eppstein, Z. Galil, G. F. Italiano and A. Nissenzweig. Sparsification - A technique for speeding up dynamic graph algorithms. In *Proceedings of the Thirty-second Symposium on Foundations of Computer Science*, 1992, IEEE.

[Fa74] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In *Complexity of Computation*, (ed. R. Karp), *SIAM-AMS Proceedings 7*, pages 27–41, 1974, SIAM-AMS.

[F83] G. F. Frederickson. Ambivalent data structures for dynamic 2-edge connectivity and k smallest spanning trees. In *Proceedings of the Thirty-second Symposium on Foundations of Computer Science*, 1991, IEEE.

[GMS93] A. Gupta, I. S. Mumick and V. S. Subrahmanian. Maintaining views incrementally. In *Proceedings of the 1993 ACM SIGMOD*, pages 157-166, 1993, ACM.

[G83] Y. Gurevich. Algebras of feasible functions. In *Proceedings of the Twenty-fourth Symposiumon Foundations of Computer Science*, pages 210–214, 1983, IEEE.

[I91] Neil Immerman. DSPACE[$n^k$] = VAR[$k+1$]. In *Proceedings of the Sixth Symposium on Structure in Complexity Theory*, pages 334-340, July 1991, IEEE.

[I89] N. Immerman. Descriptive and computational complexity. In *Computational complexity theory, ed. J. Hartmanis, Proceedings of Symposium in Applied Mathematics, 38*, pages 75–91, 1989, AMS.

[IL89] Neil Immerman and Susan Landau. The complexity of iterated multiplication. In *Proceedings of the Fourth Annual Symposium on Structure in Complexity Theory* pages 104-111, 1989, IEEE; revised version to appear in *Information and Control.*

[I87] Neil Immerman. Languages that capture complexity classes. In *SIAM Journal of Computing* **16**, *No. 4*, pages 760–778, 1987.

[J92] H. Jakobsson. On materializing views and on-line queries. In *Proceedings of International Conference on Database Theory*, Berlin, pages 407–420, October 1992.

[LST87] J. W. LLoyd, E. A. Sonenberg and R. W. Topor. Integrity constraint checking in stratified databases. In *Journal of Logic Programming, 4(4)*, pages 334–343, 1987.

[M94] J. A. Medina-Peralta. On NP-completeness under First Order Projections. In *Working paper*, 1994, Computer Science Department, University of Massachusetts, Amherst.

[MSVT93] Peter Bro Miltersen, Sairam Subramanian, Jeffrey Scott Vitter, and Roberto Tamassia. Complexity models for incremental computation. In *Preprint*, 1993.

[N82] J-M. Nicolas. Logic for improving integrity checking in relational databases. In *Acta Informatica, 18(3)*, pages 227–253, 1982.

[PY88] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *Proceedings of the Twentieth ACM Symposium on Theory of Computing*, pages 229–234, 1988, ACM.

[P] C. H. Papadimitrou. In *Computational Complexity*, Academic Press, 1994.

[P94] S. Patnaik. The dynamic complexity of approximation. In Technical Report, 1994, Computer Science Department, University of Massachusetts, Amherst.

[PI94] S. Patnaik and N. Immerman. Dyn-FO: A parallel dynamic complexity class. In Technical Report, 1994, Computer Science Department, University of Massachusetts, Amherst.

[R92] M. Rauch. Fully dynamic biconnectivity in graphs. *Proceedings of the Thirty-third Annual IEEE Symposium on Foundations of Computer Science*, 1992, IEEE.

[STV93] S. Sairam, R. Tamassia and J. Vitter. A complexity theoretic approach to incremental computation. *Proceedings of the Second Symposium on Theoretical Aspects of Computer Science*, 1993.

[U] Jeffrey D. Ullman. In *Principles of database and knowledge-base systems, Volume II*, 1989, Computer Science Press, Rockville, MD.

[Var] M. Vardi. Complexity of relational query languages. In *Proceedings of the Fourteenth Symposium on Theory of Computation*, pages 137–146, 1982, ACM.